Contents

Enabling Rate Limits using Envoy

6 minute read   page test

This task shows you how to use Envoy's native rate limiting to dynamically limit the traffic to an Istio service. In this task, you will apply a global rate-limit for the `productpage` service through ingress gateway that allows 1 requests per minute across all instances of the service. Additionally, you will apply a local rate-limit for each individual `productpage` instance that will allow 10 requests per minute. In this way, you will ensure that the `productpage` service handles a maximum of 1 request per minute through the ingress gateway, but each `productpage` instance can handle up to 10 requests per minute, allowing for any in-mesh traffic.

Before you begin

1. Setup Istio in a Kubernetes cluster by following the instructions in the Installation Guide.
2. Deploy the Bookinfo sample application.

Rate limits

Envoy supports two kinds of rate limiting: global and local. Global rate limiting uses a global gRPC rate limiting service to provide rate limiting for the entire mesh. Local rate limiting is used to limit the rate of requests per service instance. Local rate limiting can be used in conjunction with global rate limiting to reduce load on the global rate limiting service.

In this task you will configure Envoy to rate limit traffic to a specific path of a service using both global and local rate limits.

Global rate limit

Envoy can be used to set up global rate limits for your mesh. Global rate limiting in Envoy uses a gRPC API for requesting quota from a rate limiting service. A reference implementation of the API, written in Go with a Redis backend, is used below.

1. Use the following configmap to configure the reference implementation to rate limit requests to the path `/productpage` at 1 req/min and all other requests at 100 req/min.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ratelimit-config
data:
  config.yaml: |
    domain: productpage-ratelimit
    descriptors:
      - key: PATH
        value: "/productpage"
        rate_limit:
          unit: minute
          requests_per_unit: 1
      - key: PATH
        rate_limit:
          unit: minute
          requests_per_unit: 100
```

2. Create a global rate limit service which implements Envoy's rate limit service protocol. As a reference, a demo configuration can be found here, which is based on a reference implementation provided by Envoy.
3. Apply an `EnvoyFilter` to the `ingressgateway` to enable global rate

limiting using Envoy's global rate limit filter.

The first patch inserts the `envoy.filters.http.ratelimit` global envoy filter filter into the `HTTP_FILTER` chain. The `rate_limit_service` field specifies the external rate limit service, `rate_limit_cluster` in this case.

The second patch defines the `rate_limit_cluster`, which provides the endpoint location of the external rate limit service.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: filter-ratelimit
  namespace: istio-system
spec:
  workloadSelector:
    # select by label in the same namespace
    labels:
      istio: ingressgateway
  configPatches:
    # The Envoy config you want to modify
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: "envoy.filters.network.http
_connection_manager"
              subFilter:
                name: "envoy.filters.http.route
r"
      patch:
        operation: INSERT_BEFORE
        # Adds the Envoy Rate Limit Filter in H
TTP filter chain.
        value:
          name: envoy.filters.http.ratelimit
          typed_config:
            "@type": type.googleapis.com/envoy.
extensions.filters.http.ratelimit.v3.RateLimit
            # domain can be anything! Match it
to the ratelimter service config
            domain: productpage-ratelimit
            failure_mode_deny: true
            timeout: 10s
            rate_limit_service:
              grpc_service:
                envoy_grpc:
                  cluster_name: rate_limit_clus
```

```
ter
              transport_api_version: V3
    - applyTo: CLUSTER
      match:
        cluster:
          service: ratelimit.default.svc.cluste
r.local
      patch:
        operation: ADD
        # Adds the rate limit service cluster f
or rate limit service defined in step 1.
        value:
          name: rate_limit_cluster
          type: STRICT_DNS
          connect_timeout: 10s
          lb_policy: ROUND_ROBIN
          http2_protocol_options: {}
          load_assignment:
            cluster_name: rate_limit_cluster
            endpoints:
            - lb_endpoints:
              - endpoint:
                  address:
                     socket_address:
                      address: ratelimit.defaul
t.svc.cluster.local
                      port_value: 8081
EOF
```

4. Apply another `EnvoyFilter` to the

`ingressgateway` that defines the route configuration on which to rate limit. This adds rate limit actions for any route from a virtual host named `*.80`.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: filter-ratelimit-svc
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      istio: ingressgateway
  configPatches:
    - applyTo: VIRTUAL_HOST
      match:
        context: GATEWAY
        routeConfiguration:
          vhost:
            name: ""
            route:
              action: ANY
      patch:
        operation: MERGE
        # Applies the rate limit rules.
        value:
          rate_limits:
            - actions: # any actions in here
              - request_headers:
                  header_name: ":path"
                  descriptor_key: "PATH"
EOF
```

Local rate limit

Envoy supports local rate limiting of L4 connections and HTTP requests. This allows you to apply rate limits at the instance level, in the proxy itself, without calling any other service.

The following `EnvoyFilter` enables local rate limiting for any traffic through the `productpage` service. The `HTTP_FILTER` patch inserts the `envoy.filters.http.local_ratelimit` local envoy filter into the HTTP connection manager filter chain. The local rate limit filter's token bucket is configured to allow 10 requests/min. The filter is also configured to add an `x-local-rate-limit` response header to requests that are blocked.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
```

```
kind: EnvoyFilter
metadata:
  name: filter-local-ratelimit-svc
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: productpage
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: SIDECAR_INBOUND
        listener:
          filterChain:
            filter:
              name: "envoy.filters.network.http_con
nection_manager"
      patch:
        operation: INSERT_BEFORE
        value:
          name: envoy.filters.http.local_ratelimit
          typed_config:
            "@type": type.googleapis.com/udpa.type.
v1.TypedStruct
            type_url: type.googleapis.com/envoy.ext
ensions.filters.http.local_ratelimit.v3.LocalRateLi
mit
            value:
              stat_prefix: http_local_rate_limiter
              token_bucket:
                max_tokens: 10
                tokens_per_fill: 10
                fill_interval: 60s
              filter_enabled:
                runtime_key: local_rate_limit_enabl
ed
                default_value:
                  numerator: 100
                  denominator: HUNDRED
              filter_enforced:
                runtime_key: local_rate_limit_enfor
ced
                default_value:
                  numerator: 100
                  denominator: HUNDRED
              response_headers_to_add:
                - append: false
                  header:
                    key: x-local-rate-limit
                    value: 'true'
EOF
```

The above configuration applies local rate limiting to all vhosts/routes. Alternatively, you can restrict it to a specific route.

The following `EnvoyFilter` enables local rate limiting for any traffic to port 80 of the `productpage` service. Unlike the previous configuration, there is no `token_bucket` included in the `HTTP_FILTER` patch. The `token_bucket` is instead defined in the second (`HTTP_ROUTE`) patch which includes a `typed_per_filter_config` for the `envoy.filters.http.local_ratelimit` local envoy filter, for routes to virtual host `inbound|http|9080`.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: filter-local-ratelimit-svc
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: productpage
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: SIDECAR_INBOUND
        listener:
          filterChain:
```

```
          filter:
            name: "envoy.filters.network.http_con
nection_manager"
      patch:
        operation: INSERT_BEFORE
        value:
          name: envoy.filters.http.local_ratelimit
          typed_config:
            "@type": type.googleapis.com/udpa.type.
v1.TypedStruct
            type_url: type.googleapis.com/envoy.ext
ensions.filters.http.local_ratelimit.v3.LocalRateLi
mit
            value:
              stat_prefix: http_local_rate_limiter
    - applyTo: HTTP_ROUTE
      match:
        context: SIDECAR_INBOUND
        routeConfiguration:
          vhost:
            name: "inbound|http|9080"
            route:
              action: ANY
      patch:
        operation: MERGE
        value:
          typed_per_filter_config:
            envoy.filters.http.local_ratelimit:
              "@type": type.googleapis.com/udpa.typ
e.v1.TypedStruct
              type_url: type.googleapis.com/envoy.e
```

```
xtensions.filters.http.local_ratelimit.v3.LocalRate
Limit
              value:
                stat_prefix: http_local_rate_limite
r
                token_bucket:
                  max_tokens: 10
                  tokens_per_fill: 10
                  fill_interval: 60s
                filter_enabled:
                  runtime_key: local_rate_limit_ena
bled
                  default_value:
                    numerator: 100
                    denominator: HUNDRED
                filter_enforced:
                  runtime_key: local_rate_limit_enf
orced
                  default_value:
                    numerator: 100
                    denominator: HUNDRED
                response_headers_to_add:
                  - append: false
                    header:
                      key: x-local-rate-limit
                      value: 'true'
EOF
```

Verify the results

Verify global rate limit

Send traffic to the Bookinfo sample. Visit `http://$GATEWAY_URL/productpage` in your web browser or issue the following command:

```
$ curl "http://$GATEWAY_URL/productpage"
```

`$GATEWAY_URL` is the value set in the Bookinfo example.

You will see the first request go through but every following request within a minute will get a 429 response.

Verify local rate limit

Although the global rate limit at the ingress gateway limits requests to the `productpage` service at 1 req/min, the local rate limit for `productpage` instances allows 10 req/min. To confirm this, send internal `productpage` requests, from the `ratings` pod, using the following `curl` command:

```
$ kubectl exec "$(kubectl get pod -l app=ratings -o
 jsonpath='{.items[0].metadata.name}')" -c ratings
-- curl -sS productpage:9080/productpage | grep -o
"<title>.*</title>"
<title>Simple Bookstore App</title>
```

You should see no more than 10 req/min go through per `productpage` instance.